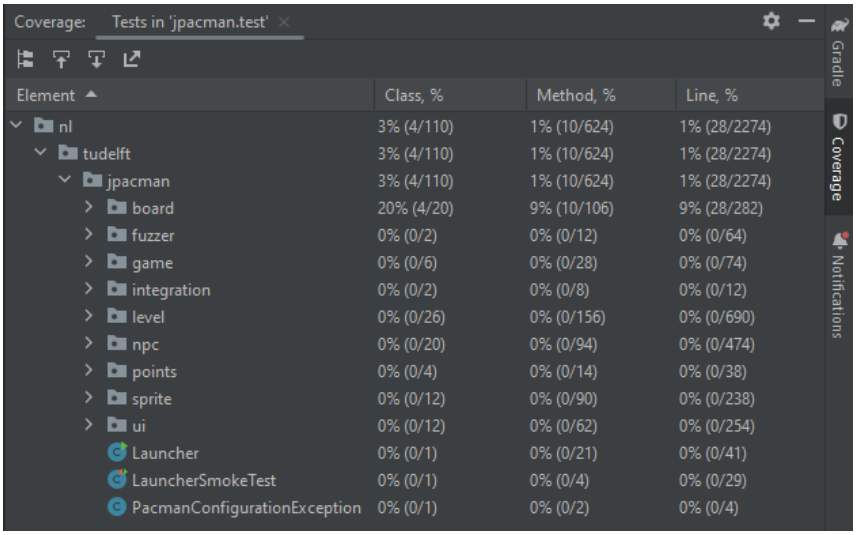
Unit Testing Report – CS 472

by Ethan Bacolod

Task 1 – JPacman Test Coverage

Initial 'jpacman [test]' with Coverage



Element	Class, %	Method, %	Line, %
nl	3% (4/110)	1% (10/624)	1% (28/2274)
tudelft	3% (4/110)	1% (10/624)	1% (28/2274)
jpacman	3% (4/110)	1% (10/624)	1% (28/2274)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	0% (0/26)	0% (0/156)	0% (0/690)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	0% (0/12)	0% (0/90)	0% (0/238)
ui	0% (0/12)	0% (0/62)	0% (0/254)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

Question:

- Is the coverage good enough?
  - This is poor coverage. I would say good coverage is greater than 70%.

Task 2 – Increasing Coverage on JPacman

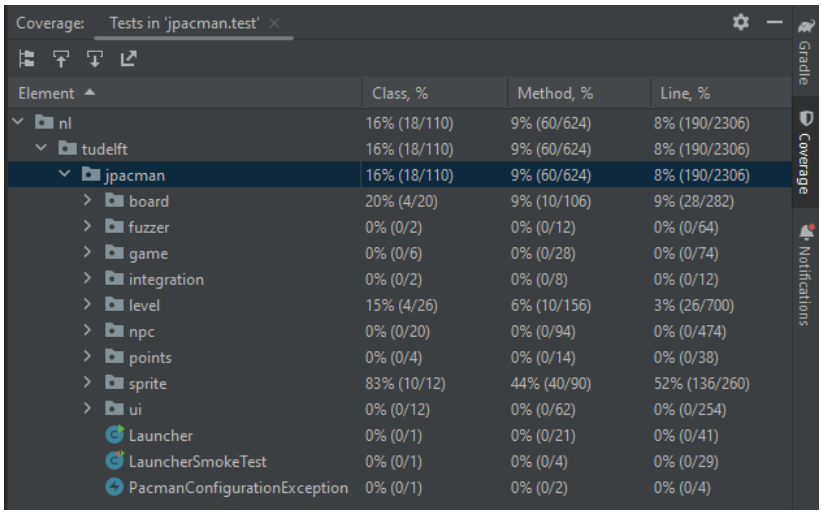
PlayerTest Code Snippet

- This code tests the Player class's isAlive() method by creating a new player. A newly created player is alive, and the method should return true.

```
public class PlayerTest {

    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    1 usage
    private Player ThePlayer = Factory.createPacMan();

    no usages  new *
    @Test
    void testAlive() { assertThat(ThePlayer.isAlive()).isEqualTo( expected: true); }
}
```

PlayerTest Coverage

- +14 Classes (+13%); +50 Methods (+8%); +162 Lines (+9%)



Task 2.1

**NOTE: Each unit test in Task 2.1 includes the PlayerTest.*

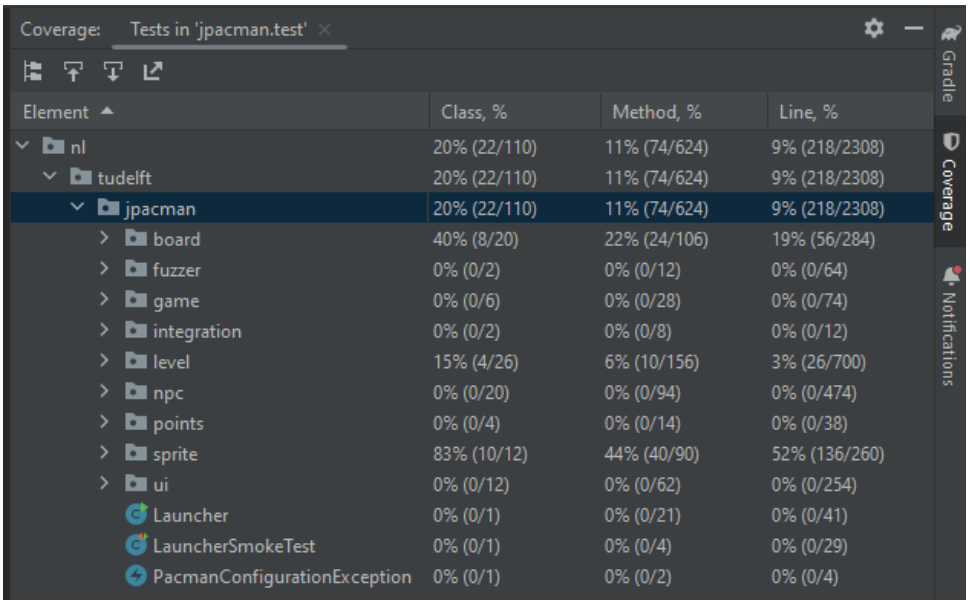
WithinBoardTest Code Snippet (Board Class)

- This snippet tests the Board class's withinBoarders() function. It creates a board and checks if the coordinates are within the board.

```
no usages  new *
public class WithinBoardTest {

    1 usage
    private static final int MAX_WIDTH = 5;
    1 usage
    private static final int MAX_HEIGHT = 4;

    1 usage
    private final Square[][] grid = {
        { mock(Square.class), mock(Square.class), mock(Square.class) , mock(Square.class), },
        { mock(Square.class), mock(Square.class), mock(Square.class) , mock(Square.class), },
        { mock(Square.class), mock(Square.class), mock(Square.class) , mock(Square.class), },
        { mock(Square.class), mock(Square.class), mock(Square.class) , mock(Square.class), },
        { mock(Square.class), mock(Square.class), mock(Square.class) , mock(Square.class), }
    };
    1 usage
    private final Board board = new Board(grid);

    no usages  new *
    @Test
    void withinBoardTest(){
        for(int x = 0; x < MAX_WIDTH; x++) {
            for (int y = 0; y < MAX_HEIGHT; y++) {
                assertThat(board.withinBorders(x, y)).isTrue();
            }
        }
    }
}
```

WithinBoardTest Coverage (Board Class)

- +4 Classes (+4%); +14 Methods (+2%); +28 Lines (+1%)



Element	Class, %	Method, %	Line, %
nl	20% (22/110)	11% (74/624)	9% (218/2308)
tudelft	20% (22/110)	11% (74/624)	9% (218/2308)
jpacman	20% (22/110)	11% (74/624)	9% (218/2308)
board	40% (8/20)	22% (24/106)	19% (56/284)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	15% (4/26)	6% (10/156)	3% (26/700)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	44% (40/90)	52% (136/260)
ui	0% (0/12)	0% (0/62)	0% (0/254)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

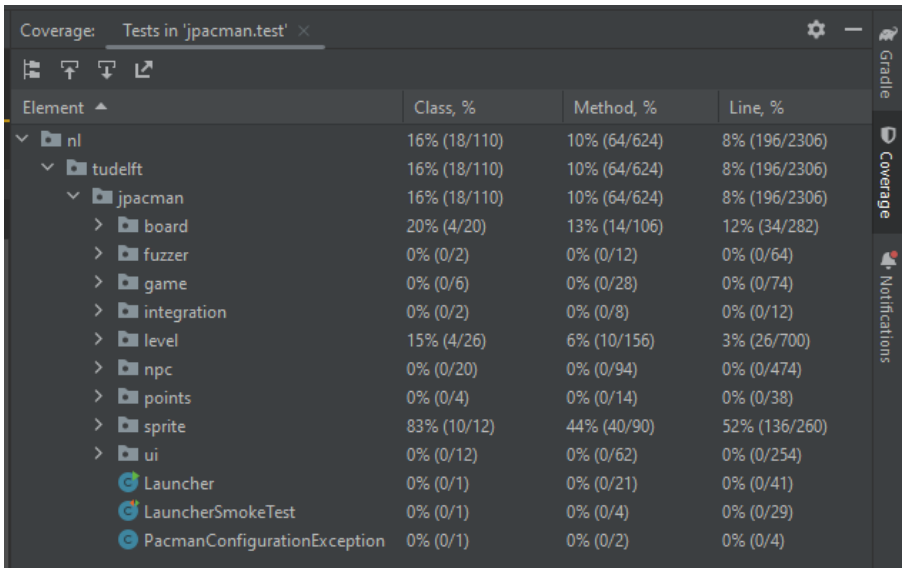
GetDirectionTest Code Snippet (Unit Class)

- This section tests the Unit class's getDirection() method. This is done by assigning a value to the unit and seeing if we get the same value back.

```java
package nl.tudelft.jpacman.board;

import ...

no usages  new *
public class UnitClassTest {
    3 usages
    private Unit unit;

    no usages  new *
    @BeforeEach
    public void setUp() {
        unit = new BasicUnit();
        unit.setDirection(Direction.SOUTH);
    }

    no usages  new *
    @Test
    void getDirectionTest() { assertThat(unit.getDirection()).isEqualTo(Direction.SOUTH); }

}
```

GetDirectionTest Coverage (Unit Class)

- +4 Classes (+4%); +4 Methods (+1%); +6 Lines (+0%)



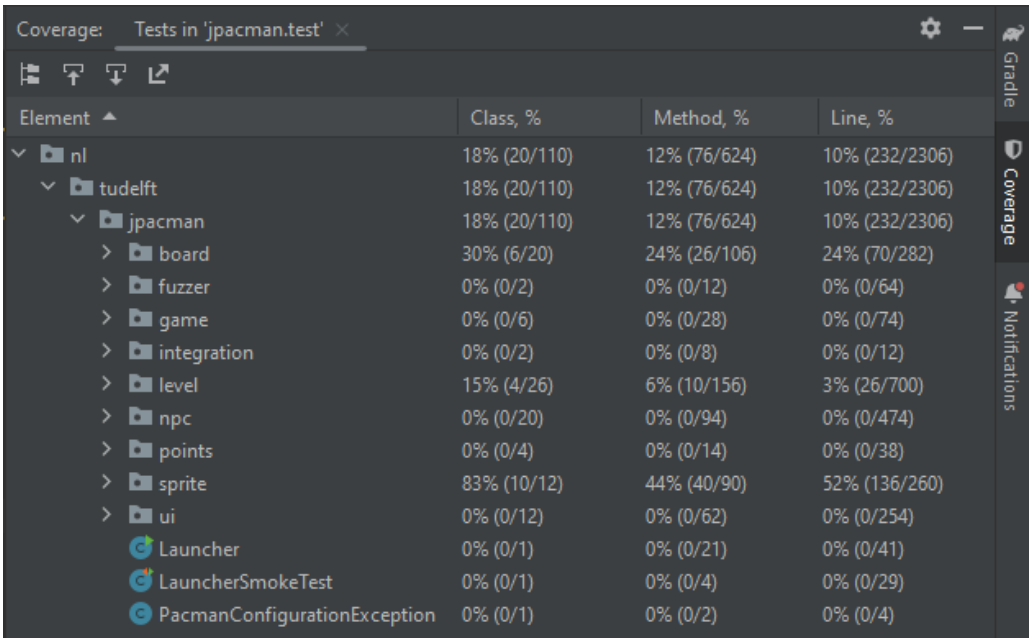
HasSquareTest Code Snippet (Unit Class)

- This code tests the Unit class's hasSquare() method. A unit is created with its squared occupied. When occupied, the function should return true.

```
no usages  new *
public class UnitClassTest {
    2 usages
    private Square square;
    3 usages
    private Unit unit;

    no usages  new *
    @BeforeEach
    public void setUp() {
        square = new BasicSquare();
        unit = new BasicUnit();
        unit.occupy(square);
    }

    no usages  new *
    @Test
    void hasSquareTest(){
        assertThat(unit.hasSquare()).isTrue();
    }

}
```

HasSquareTest Coverage (Unit Class)

- +2 Classes (+2%); +16 Methods (+3%); +42 Lines (+2%)



Task 3 – JaCoCo Report on JPacman

Questions:

- Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?
  - To me, they look a little similar but there are differences. I can see the number of methods and lines that were missed for each piece of software, but the numbers are completely different. The level package on IntelliJ has 156 methods, but JaCoCo has 69 in total.
- Did you find helpful the source code visualization from JaCoCo on uncovered branches?
  - I thought the color visualization on the source code was useful. It's very useful for picking out which lines have been tested and which ones have not.
- Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?
  - I do like JaCoCo's due to the fact that it gives percentage bars as well as highlights on the source code itself.

Link To Team Forked Repo: https://github.com/ebacolod/cs472-team6

Link To JPacman Forked Repo: https://github.com/ebacolod/jpacman/tree/jpacman